Tests fonctionnels : TP Selenium

I) Utiliser Selenium IDE

1) Introduction

Selenium IDE est un environnement de développement intégré pour les tests Selenium. On le trouve sous forme d'extension pour **Firefox**. Il permet d'enregistrer, d'éditer et déboguer des tests mais pas seulement, car en sa qualité d'IDE il vous permettra aussi de modifier vos scripts à la main.

Caractéristiques :

- Facilité d'enregistrement et de lecture
- Reconnaissance de sélection des identifiants, les noms, ou, au besoin XPath
- Pas à pas à travers les tests
- Sauvegarde des tests au format HTML, Java, Ruby scripts, …

2) Mode d'emploi

Après avoir installé le plugin Selenium IDE (aidez-vous du tutoriel suivant http://www.objis.com/formation-java/tutoriel-selenium-installation-selenium-ide.html), il suffit de procéder de la façon suivante :

- Lancer Firefox
- Menu Outils > Selenium IDE
- Vérifier que le bouton d'enregistrement (bouton rond rouge) est bien "enfoncé"
- Dans votre fenêtre Firefox, taper l'URL de la page à tester
- Exécuter les différentes étapes de votre scénario
- Une fois votre scénario fini, stopper l'enregistrement en cliquant sur le bouton d'enregistrement (record)
- Sauvegarder votre test (au format HTML)
- Lancer le pour valider son bon fonctionnement (en cliquant sur la flèche verte)

Si besoin, compléter votre test :

- Ajouter des commandes "pause" si Selenium va plus vite que votre application ou détecte mal la fin de chargement de la page
- Ajouter les commandes Selenium nécessaires si certaines parties de votre test n'ont pas pu être enregistrées par Selenium IDE (cas de certains menus/formulaire avec du javascript)
- ...

3) Documentation

Vous trouverez une documentation complète à l'adresse http://docs.seleniumhq.org/docs/

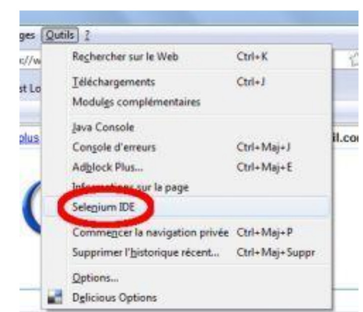
4) Application

☞ **Installer le pluggin Selenium IDE**
☞ **Réaliser un test du moteur de recherche de Google**

Le test s'assure que le site de Selenium (seleniumhq.org) fait partie des résultats de la première page lorsqu'on lance une recherche **Google** sur "selenium ide" puis lance la page Downloads selenium.

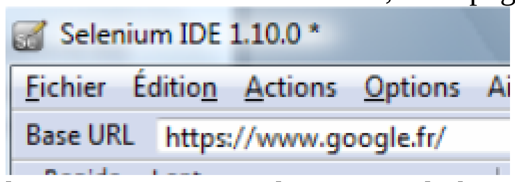
☞ Enregistrez votre séquence :

1. Lancez Firefox
2. Allez sur la page d'accueil de Google
3. Lancez Selenium IDE à partir de Firefox, dans le menu *Outils :*



Remarque : Il est pratique de voir la fenêtre de Selenium mise à jour au fur et à mesure de l'enregistrement du test. Pour cela, disposez Firefox et Selenium IDE de sorte à voir les deux fenêtres simultanément.

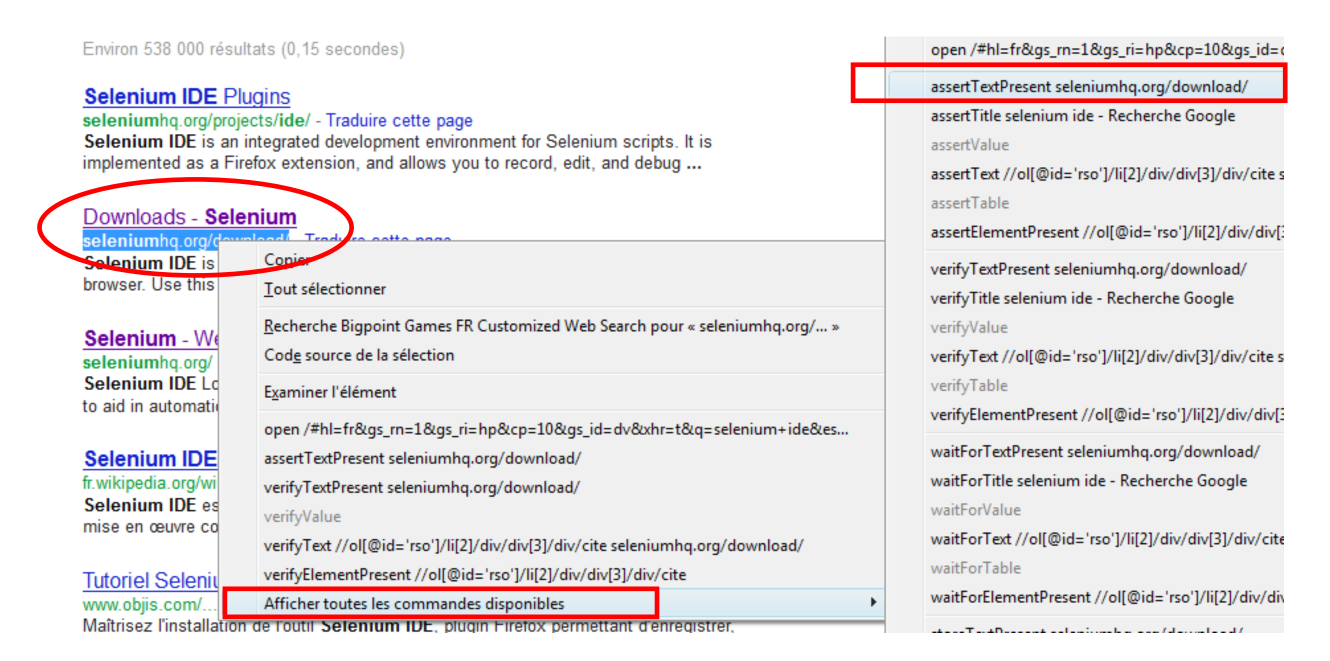
4. Saisissez l'URL de la page d'accueil de **Google** dans le champ ***Base URL*** de Selenium IDE. En effet, cette page sera le point de départ de notre test.



5. Lancez l'enregistrement en cliquant sur le bouton ***Record*** de Selenium IDE. Selenium est maintenant en train d'enregistrer les actions de l'utilisateur dans Firefox : click, saisie...

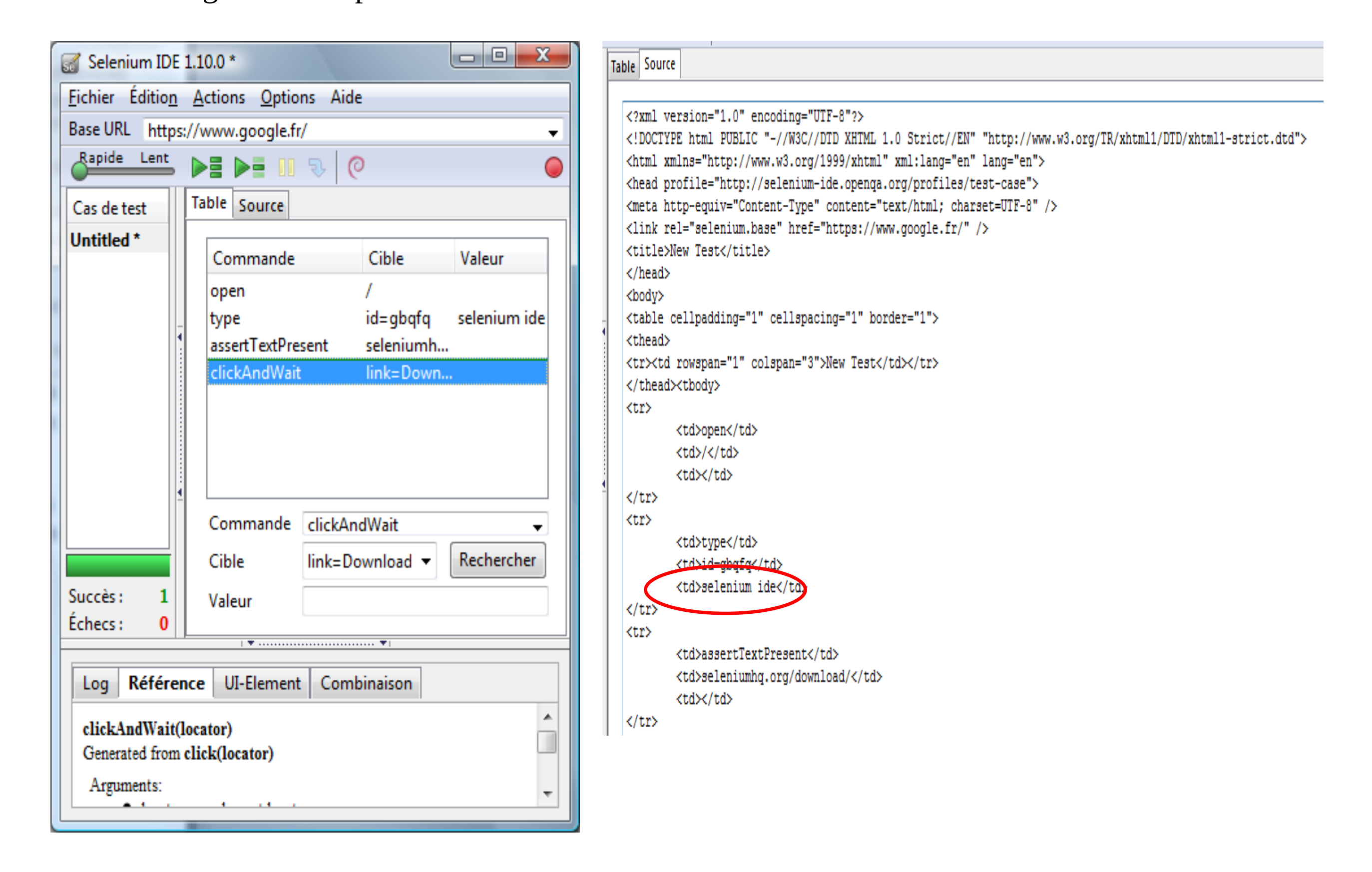


6. Jouez la séquence à enregistrer : Saisissez "selenium ide" dans la page de Google et cliquez sur *Recherche Google*. On constate que Selenium IDE liste les actions au fur et à mesure.
7. Ajoutez une **assertion** : Dans la page de résultat de **Google**, nous voulons nous assurer qu'un lien vers le site de téléchargement Selenium est présent. Il apparait effectivement, on peut voir le texte "seleniumhq.org/download" qui apparait au bas du deuxième résultat. Sélectionnez ce texte dans Firefox, faites un clic droit et sélectionnez *Afficher toutes les commandes puis assertTextPresent seleniumhq.org/download*



8. Stoppez l'enregistrement.

9. Vérifiez que Selenium a enregistré la séquence que vous venez de jouer.
   L'onglet *Source* permet d'accéder au test sous la forme HTML :



Le test est modifiable aussi bien dans l'onglet *Table* que *Source*.

<u>Remarque:</u>

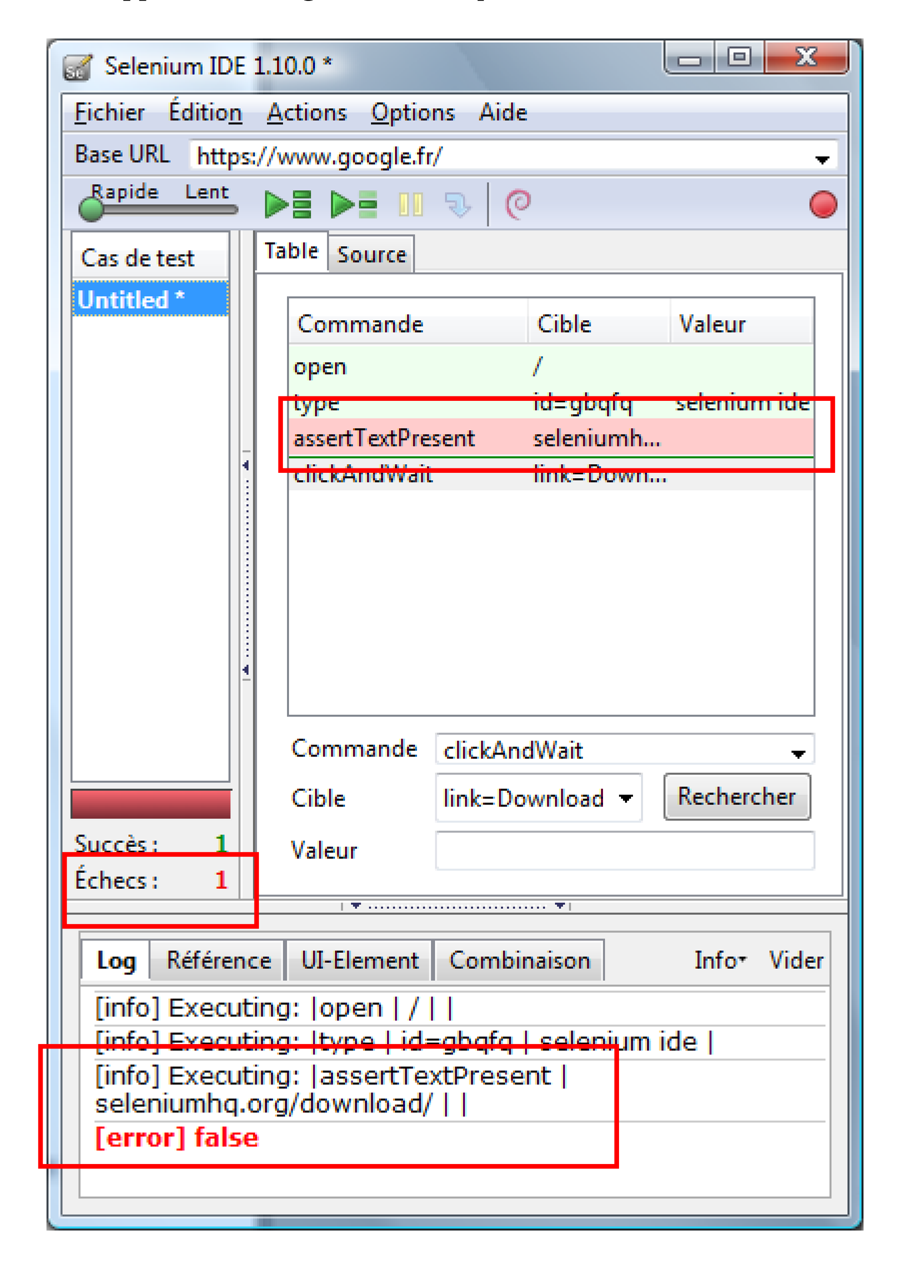
La commande type permet de simuler la saisie d'une valeur dans un champ. Vous remarquez que la cible est désignée par 'id=gbqfq' et la valeur saisie sera 'selenium ide'.

Si dans votre navigateur, sur la page de recherche google, vous placez votre curseur dans la zone de recherche, et que vous faites un clic droit, puis *Inspect element with FireBug* vous verrez apparaître le code HTML de cette partie de votre page et vous retrouverez l'identifiant de cet élément:

```
<input id="gbqfq" class="gbqfif" type="text" value="" autocomplete="off" n
ame="q" style="border: medium none; padding: 0px; margin: 0px; height:
auto; width: 100%; background: url("data:image/gif;base64,R0lGODlhAQABAID
/AMDAwAAAACH5BAEAAAAALAAAAAABAAEAAAICRAEAOw%3D%3D") repeat scroll 0% 0%
transparent; position: absolute; z-index: 6; left: 0px; outline: medium
none;" dir="ltr" spellcheck="false">
```

10. Rejouez le test. Maintenant que Selenium a enregistré la séquence, vous pouvez la rejouer à loisir. Selenium IDE joue le test "pour de vrai" en pilotant Firefox. Vous devez voir ce dernier aller sur la page d'accueil de **Google**, saisir "selenium ide" et charger la page de résultat. Selenium IDE liste ce qu'il a exécuté dans l'onglet *Log*. Que constatez-vous ?

L'assertion a été évaluée mais le test a échoué comme signalé dans l'onglet *Log*. De plus la ligne du test apparait en rouge afin d'indiquer l'échec.

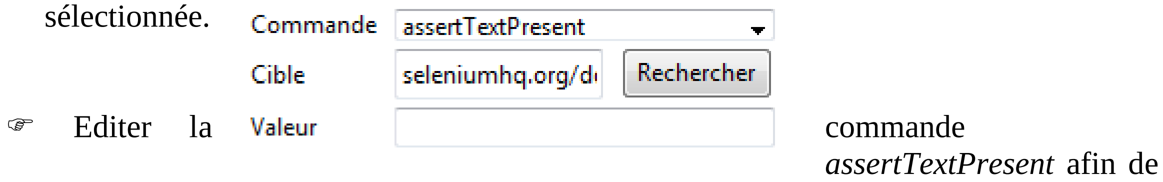


Quel est le problème ?

Le problème se pose quand Selenium joue plus vite que le navigateur (le cas se pose par exemple quand un site utilise la technologie Ajax). Il faut donc demander à Selenium d'attendre la réponse. Pour cela ajouter une ligne de commande *waitForTextPresent.*
Cette ligne de commande ressemblera à la commande *assertTextPresent* que vous pouvez éditer pour vous aider.

Pour éditer une commande :

- Cliquez sur la ligne correspondant à la commande.
- En bas de la liste des commandes, il y a un panneau avec trois champs : **Command, Target et Value**. Ils permettent d'éditer les paramètres de la commande actuellement sélectionnée.

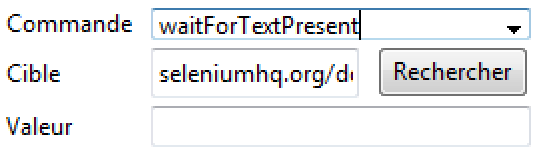


☞ Editer la commande *assertTextPresent* afin de réaliser un copier de la cible

Pour ajouter une commande :

- Menu Edition de Selenium IDE > Ajouter une commande : ajoute une commande avant la commande sélectionnée.

☞Ajoutez la nouvelle commande «*waitForTextPresent*» avant la commande *assertTextPresent.*



Vous devez obtenir :

Commande	Cible	Valeur
open	/	
type	id=gbqfq	selenium ide
waitForTextPresent	seleniumh...	
assertTextPresent	seleniumh...	
clickAndWait	link=Down...	

☞Rejouez le test. Cette fois le test doit se dérouler correctement jusqu'à la dernière commande.

☞Enregistrez votre test !

5) Choisir entre une commande VERIFY et ASSERT :

Certaines commandes commencent par « **assert** » et d'autres par « **verify** ».
« **assert** » stoppe l'exécution du script si une erreur survient. Utile pour vérifier que l'on est sur la bonne page, ou qu'une étape critique a bien été franchie. C'était le cas dans votre test car le chargement de la page souhaitée ne peut se faire que si le lien est bien présent sur la page en cours !

« **verify** » va tracer toutes les erreurs s'il y en a, mais le script va continuer son exécution. Utile pour vérifier les valeurs des saisies lorsqu’elles sont nombreuses sur une page.

6) Compléter votre test:

☞Ajoutez une commande permettant de vérifiez que le titre de la page chargée après un clic sur le lien Selenium-Downloads (commande **clickAndWait)** est bien "Donwloads" (regardez dans la liste des commandes *assert*)

☞Vérifiez que votre test s'exécute bien.

☞Ajoutez les commandes nécessaires à l'affichage de la page '**Selenium IDE - Release Notes**' si l'élément **link=Release Notes** est bien présent.

☞Vérifiez que votre test s'exécute bien.

☞Glissez des erreurs dans votre test à différents endroits (en plusieurs fois !) et vérifiez que votre test s'arrête si la vérification est effectuée avec un ***assert*** et qu'il continue son exécution après une erreur signalée lors de l'exécution d'une commande ***verify***…

7) Suite de tests

Avec Selenium IDE vous pouvez créer une suite de tests qui constitue un regroupement de tests.

II) Intégration

Pour le moment, nous avons un test que nous pouvons jouer à partir d'une interface graphique.

C'est un début, mais il faut aller plus loin :

Exécution dans d'autres navigateurs : Développer avec Firefox pourquoi pas, mais il faut s'assurer que l'application fonctionne aussi avec Internet Explorer, Chrome, Safari et les autres.

Gérer de nombreux tests : Dans un cas réel, on a plusieurs dizaines, centaines voir milliers de tests. Cette interface graphique n'est pas viable pour un tel usage.

Tests intégrés : Jouer le test à partir de l'interface graphique est pratique lorsqu'on le met au point. En revanche, cela n'est plus praticable lorsqu'on veut jouer le test au même titre que les autres tests de l'application (tests unitaires, etc.). Les tests générés avec Selenium doivent pouvoir être lancés dans le cadre d'une intégration continue.

Modification et déclinaison du test : Nous pourrions souhaiter décliner notre test de plusieurs façons. Par exemple, en recherchant différents termes. Avec Selenium IDE, notre seule option est de jouer le scénario encore et encore pour créer nos tests. Cela va vite devenir ennuyeux.

<u>NB</u> : Ne pas tester cette partie orientée SLAM 4, la lire pour info seulement et passez au TP n°2 qui valorisera le travail présenté à l'épreuve E4.

1) Exemple : intégration d'un test dans un environnement JAVA.

L'étape d'intégration est très rapide car Selenium génère directement un test **JUnit** pour vous :

- **Démarche :**

Dans Selenium IDE, sélectionnez :

Fichier > Exporter le Test sous… > Java / JUnit 4 /Remote Control

Le choix de Remote Control permet d'utiliser le serveur Selenium RC pour jouer les tests JUnit. Ici j'ai nommé le fichier testEnJava.

Selenium génère alors votre test au format JUnit. On reconnait les différentes étapes, mais "à la Java" : **selenium.open("/")** pour ouvrir une page, etc.

Contenu de mon fichier :

```
package com.example.tests;

import com.thoughtworks.selenium.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;
import java.util.regex.Pattern;

public class testEnJava {
	private Selenium selenium;

	@Before
	public void setUp() throws Exception {
		selenium = new DefaultSelenium("localhost", 4444, "*chrome",
"https://www.google.fr/");
		selenium.start();
	}

	@Test
	public void testTestEnJava() throws Exception {
		selenium.open("/");
		selenium.type("id=gbqfq", "selenium ide");
		for (int second = 0;; second++) {
			if (second >= 60) fail("timeout");
			try { if (selenium.isTextPresent("seleniumhq.org/download/")) break; }
catch (Exception e) {}
			Thread.sleep(1000);
		}

		assertTrue(selenium.isTextPresent("seleniumhq.org/download/"));
		selenium.click("link=Downloads - Selenium");
		selenium.waitForPageToLoad("30000");
		assertEquals("Downloads", selenium.getTitle());
		for (int second = 0;; second++) {
			if (second >= 60) fail("timeout");
			try { if (selenium.isElementPresent("link=Release Notes")) break; }
catch (Exception e) {}
			Thread.sleep(1000);
		}

		assertTrue(selenium.isElementPresent("link=Release Notes"));
		selenium.click("link=Release Notes");
		selenium.waitForPageToLoad("30000");
	}

	@After
	public void tearDown() throws Exception {
		selenium.stop();
	}
```

}

Rappel :

La méthode **setUp**() est une méthode qui est systématiquement exécutée avant les autres méthodes de test. Elle va nous permettre d'implémenter un objet Selenium et de démarrer une session du navigateur souhaité (ici **chrome* correspond à Firefox et non à Google Chrome).

La méthode **tearDown**() est une méthode qui est systématiquement exécutée après l'exécution de toutes les autres méthodes de test. Elle permet de fermer la session du navigateur.

Remarque : Pour plus de détails sur les méthodes disponibles, consultez la documentation JAVA.

Pour jouer le test, il est nécessaire d'avoir lancé le **serveur Selenium RC**.
C'est un fichier **jar (exemple selenium-server-standalone-2.30.0.jar)**, pour le lancer il suffit de taper la commande java depuis la fenêtre DOS
Par exemple **java -jar selenium-server-standalone-2.30.0.jar**

- soit après vous être positionné dans le répertoire où vous avez enregistré le fichier jar,
- soit après avoir défini son chemin dans la variable d'environnement CLASSPATH.

Travail à faire :

☞Créez un nouveau projet ***java application*** sous Eclipse que vous nommerez testSelenium.

☞Créez un Test **JUnit** nommé testEnJava (même nom que le test à exporter au format JUnit4) à votre projet (package par défaut).

☞Ajoutez le fichier *jar* **selenium-server-standalone-2.30.0.jar** au Build Path de votre projet (menu Projet > Properties > Java Build Path > onglet Librairies>Add external Jars)**.**

☞Si ce n'est pas déjà fait, exportez votre test de recherche google au format **JUnit 4 RC** (depuis l'IDE de Selenium) sous le nom testEnJava.

☞Copiez le contenu de ce fichier (sauf la référence au package) dans la classe du Test JUnit créé.

☞Vérifiez que votre classe hérite de la classe **SeleneseTestBase** (Cela est nécessaire pour la méthode verifyEquals). Complétez au besoin le code de la classe.

☞Retrouvez la (ou les) instruction(s) correspondant à chaque commande. (Indiquez-les en commentaires.

☞Démarrez le serveur **Selenium RC.**

☞Exécutez votre fichier en tant que Junit Test, et vérifiez que tout se passe bien. (il peut être "pratique" de mettre en commentaire la ligne 'selenium.stop', faites l'essai avec et sans)